Instrumente si Tehnici de Baza in Informatica

Semestrul I 2024-2025

Vlad Olaru

Curs 4 - outline

- fisiere si directoare (epilog)
- procese
- inter-process communication
- semnale

Sumar comenzi utile pt. lucrul cu fisiere

- *mkdir* – creeaza directoare
- *rmdir* – sterge directoare (cu conditia sa fie goale)
- *touch* – creeaza un fisier gol daca nu exista deja
- *mv* – muta directoare sau fisiere
  - redenumeste, nu se face copiere fizica decat daca datele se muta de pe un disc pe altul
- *cp* – copiaza (fizic, duplica) directoare (cu –r) sau fisiere
  - se pot copia directoare/fisiere multiple intr-un director destinatie

    $ cp <fisier-1> <fisier-2> …. <fisier-n> <director>
- *rm* – sterge directoare (cu –r) sau fisiere
- *ln* – creeaza linkuri hard sau simbolice
- *mknod* – creeaza fisiere speciale tip caracter, bloc sau FIFO

  $ mknod /dev/sda3 b 8 3

Wildcards

- caractere speciale interpretate de shell
- ^ – simbolizeaza inceputul liniei
- $ – simbolizeaza sfarsitul liniei
- atom = caracter sau set de caractere grupat cu [ ] sau ( )
- * – 0 sau mai multe aparitii ale atomului precedent
- + – cel putin 1 aparitie a atomului precedent, posibil mai multe
- ? – cel mult o aparitie a atomului precedent (0 sau 1 aparitii)

Cautarea in fisiere

- comanda uzuala: *grep*
- foloseste tipare si expresii regulate
- synopsis

  $ grep <expresie> <fisiere>
- optiuni utile

-R / -r	cautare recursiva (cu/fara dereferentiere linkuri simbolice)
-i	case insensitive
-n	tipareste nr liniei pe care s-a gasit expresia
-w	cauta cu exactitate tiparul furnizat (cuvant, word)
-v	inverseaza sensul matching-ului (afiseaza liniile care nu se potrivesc)

- ex: *$ grep printf *.c*

  *$ grep –v –w printf *.[ch]*

  $ ls –l | grep ^d

Cautarea in directoare

- comanda uzuala: *find*
- synopsis:

  $ find <pathname> -name <expresie>

- optiuni utile

  -exec executa comanda specificata asupra elementelor gasite

  -type limiteaza rezultatele afisate la un anumit tip (eg, fisiere)

  -iname similar cu –name dar case insensitive

  -maxdepth limiteaza nivelul de recursivitate

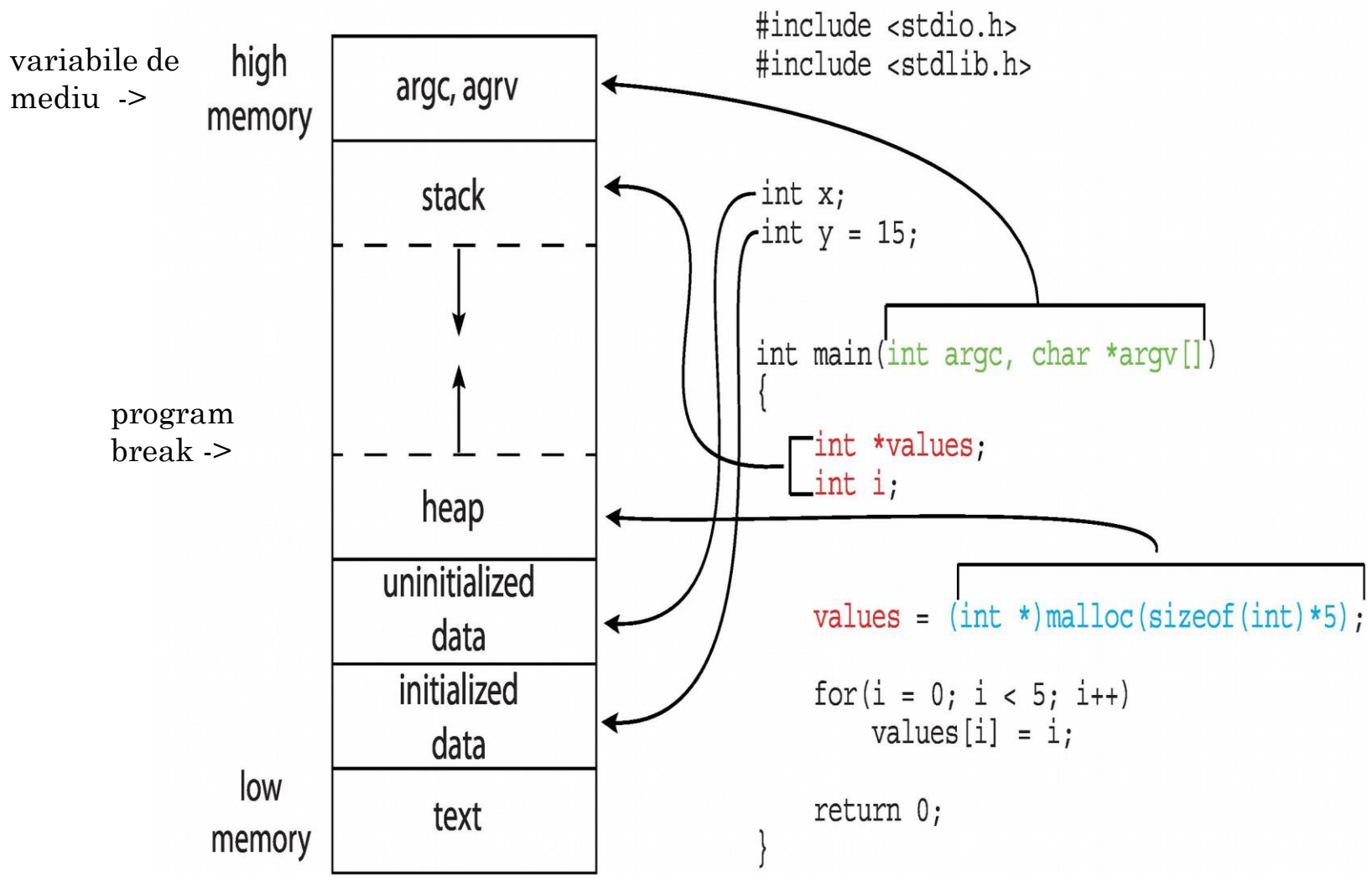
- ex: *$ find / -name hello*

  *$ find . –iname *.o –exec rm {} \;*

Procese

- proces = abstractia executiei unui program (instr + date)
  - in fapt, este programul + starea executiei sale la un moment dat (reflectata de registrele CPU si valorile variabilelor din program)
  - identificat prin *Process ID (PID)*
  - comanda *ps* afiseaza PID-urilor proceselor aflate in rulare momentan
  - mai multe instante in rulare ale aceluiasi program sunt procese diferite, cu PID-uri diferite
- executia proceselor este secventiala, nu exista executie paralela a instructiunilor intr-un singur proces
- mai multe parti
  - codul program, cunoscut si ca sectiunea de *text*
  - starea curenta reflectata de registrele CPU
  - *stiva* contine date temporare
    - parametrii functiilor, adrese de retur, variabile locale
  - sectiunile de *date initializate/neinitializate*
    - contin datele globale
  - *heap-ul* contine memoria alocata dinamic de catre program in timpul executiei

Imaginea unui program C in memorie



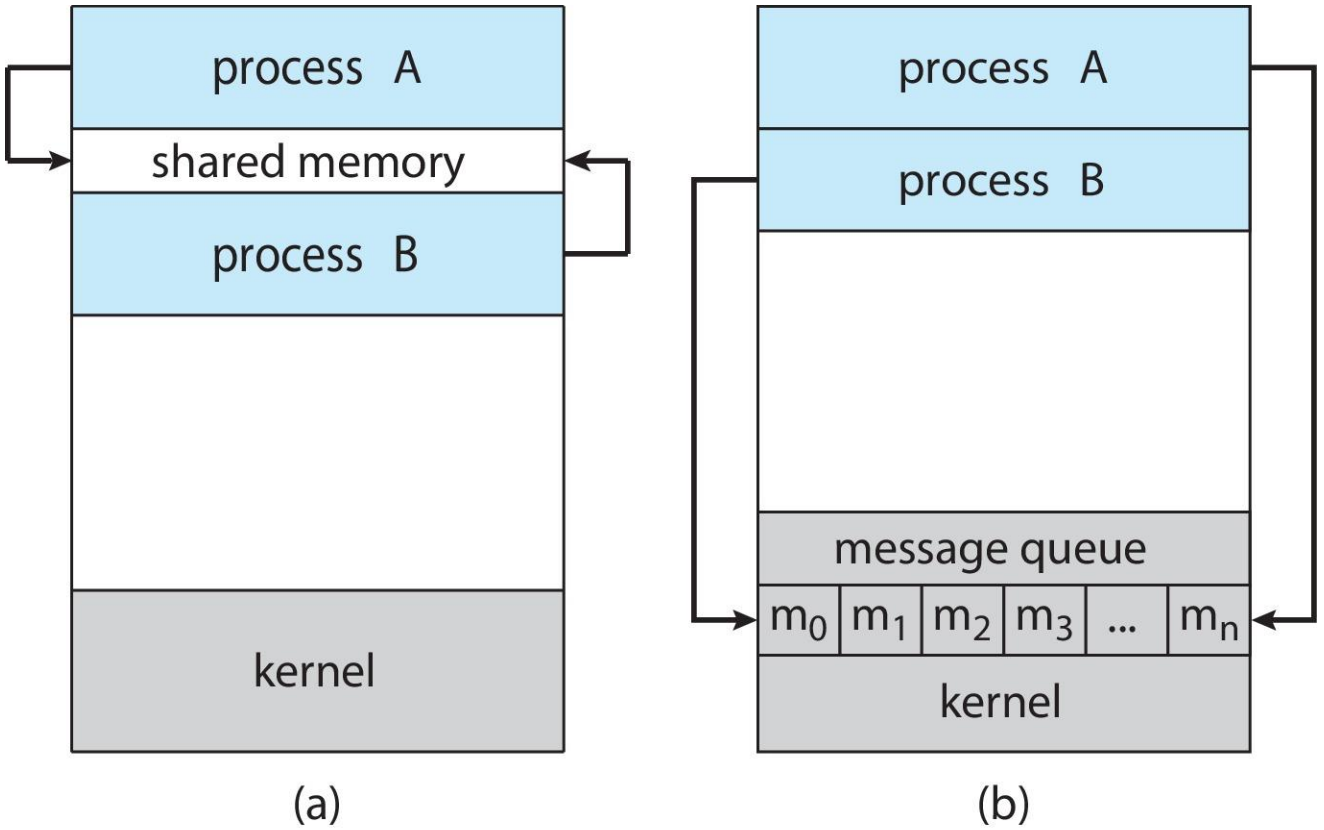
Comunicare inter-procese (IPC)

- procesele pot fi ***independente*** **sau** ***cooperante***
- procesele cooperante pot afecta sau pot fi afectate de alte procese, inclusiv prin partajarea datelor
- motive de cooperare
  - partajarea informatiei
  - accelerarea calculului
  - modularitate
  - confort
- procesele cooperante necesite mijloace de comunicare inter-proces (IPC)
- modele IPC
  - memorie partajata
  - schimb de mesaje (message passing)

Modele de comunicare

(a) Memorie partajata.

(b) Message passing.



IPC – Message Passing

- procesele comunica intre ele fara variabile partajate
- operatii:
  - **`send`**(*message*)
  - **`receive`**(*message*)
- dimensiunea mesajului este fixa sau variabila

Tipuri de comunicare

- *directa*: procesele isi folosesc identitatea explicit
  - **`send`** (*P, message*) – trimite un mesaj procesului P
  - **`receive`**(*Q, message*) – primeste un mesaj de la procesul Q
  - ex: *sockets* (canale de comunicatie TCP/IP)
- *indirecta*: mesajele sunt trimise si receptionate in/din casute postale (numite si porturi)
  - fiecare casuta are un ID unic
  - procesele pot comunica doar daca partajeaza o casuta
  - operatii
    - creeaza casuta (port)
    - trimite si primeste mesaje prin intermediul casutei postale
    - sterge casuta
  - primitive de comunicare
    - **`send`**(*A, message*) – trimite mesaj in casuta postala A
    - **`receive`**(*A, message*) – primeste mesaj din casuta postala A

Sincronizare

- schimbul de mesaje poate fi **blocant** sau **neblocant**
- **modul blocant** este considerat **sincron**
  - **send blocant** – transmitatorul e blocat pana cand se primeste mesajul
  - **receive blocant** – receptorul e blocat pana cand un mesaj e disponibil
- **modul neblocant** este considerat **asincron**
  - **send neblocant** – transmitatorul trimite mesajul si continua
  - **receive neblocant** – receptorul primeste:
    - un mesaj valid, sau
    - mesaj vid (null)
- sunt posibile diferite combinatii
  - daca *send* si *receive* sunt ambele blocante, avem un **rendezvous**

Comunicare FIFO in Unix

- prin *pipe-uri* anonime sau cu nume
- *pipe-uri anonime*
  - combinatie de comunicare directa/indirecta
  - procesele cooperante isi cunosc implicit identitatea
  - comunicare unidirectionala

    $ cat hello.c | grep printf
- *pipe-uri cu nume (FIFOs)*
  - fisiere speciale cu nume
  - comunicare indirecta, procesele cooperante nu isi cunosc identitatea
  - create cu comenzile *mknod/mkfifo*
  - odata create, folosite la fel ca fisierele, dar cu politica de acces FIFO

```
$ mknod myfifo p              # ⇔ mkfifo myfifo
$ cat < myfifo &              # trebuie sa existe mai intai un
                              # receptor care asteapta date
$ echo "write some message in myfifo" > myfifo
```

Semnale

- notificari asincrone ale procesului referitoare la producerea anumitor evenimente
  - echivalentul software al exceptiilor (HW sau SW)
  - se trimit fie intre procese, fie de catre kernel catre un proces
  - generate din mai multe surse:
    - apel sistem *kill(pid, semnal)*
    - comanda *kill:* kill –TERM <pid>
    - tastatura: DEL, Ctrl-C, Ctrl-Z, etc
    - anumite evenimente soft si hard generate de kernel
  - ex:
    - terminarea unui proces notificata asincron parintelui cu SIGCHLD
    - procesul care executa o impartire la zero primeste un semnal SIGFPE
    - accesul ilegal la memorie (eg, memorie nealocata) genereaza SIGSEGV
- reactia procesului la primirea unui semnal depinde de
  - tipul semnalului
  - decizia programului de a trata sau nu evenimentul semnalat

Posibilitati de tratare a semnalelor

1) semnalul e ignorat

- specific pt. evenimente care nu rezulta in erori/consecinte majore

2) terminarea programului (semnalul “ucide” procesul)

- valabil pt. restul evenimentelor

3) tratarea semnalului cf. indicatiei/dispozitiei programului, despre care se spune ca “prinde” semnalul

- se face cu ajutorul unei rutine de tratare a semnalului (handler)

Obs: nu orice semnal poate fi “prins” !

Ex: SIGKILL/SIGSTOP nu pot fi prinse, KILL termina invariabil programul

$ kill -KILL <pid> # ⇔ kill -9 <pid>

Exemple utilizare semnale

- comanda *kill*
  - foloseste un PID identificat in prealabil cu comanda *ps*

    $ kill –<nume-semnal/nr-semnal> <PID>

    $ kill –TERM 4899 *# ⇔ kill 4899 sau kill -15 4899*

  Obs: numele semnalului poate fi complet, eg, SIGTERM sau prescurtat, eg TERM

    $ kill –l *# afiseaza toate semnalele disponibile*

    $ kill –l 15 *# mapeaza nr de semnal in nume*

- semnalele pot fi generate voluntar de catre utilizator
  - ex: *Ctrl-c* genereaza SIGINT
    - uzual termina procesul rulat de shell
    - daca procesul prinde SIGINT, la apasarea *Ctrl-c* se executa signal handlerul asociat SIGINT de catre proces (i.e., procesul nu moare automat)

      $ kill –SIGINT 4899 *# ⇔ kill -2 4899*
  - *Ctrl-z* genereaza SIGTSTP
    - suspenda executia procesului curent (poate fi continuat cu comenzi tip job control, *fg/bg*)

      $ kill –SIGTSTP 4899 *# ⇔ kill -20 4899*